# CPO-SQL: Boosting Small LLMs for Text-to-SQL via Efficient In-Context Learning and Preference Optimization

**Anonymous ACL submission**


## Abstract

Most recent researches in Text-to-SQL parsing overly rely on the proprietary Large Language Models (LLMs), raising concerns of data privacy and inference overheads. To narrow the gap between small LLMs and proprietary LLMs in Text-to-SQL, we introduce CPO-SQL, an approach aiming to efficiently boost the capability of small LLMs via In-**C**ontext Learning and **P**reference **O**ptimization. This approach builds the enhanced training set by sampling demonstrations from beta distribution based on the similarity of questions and SQL, and then fine-tune the small LLMs to empower them with ICL capabilities of Text-to-SQL. Further, we propose a new Spider preference set, constructed by an agile semi-automated process, based on six types of SQL optimization. On this basis, we employ SFT-enhanced preference optimization to support the mixed training on the supervised set and the preference set, enabling us to optimize the SQL generation in complex query scenarios while maintaining the learning of original data. By this way, we can balance the generation ability of small LLMs for questions of varying difficulty. Finally, we evaluate our method on Spider and its three robustness-diagnostic variants, shedding light on the strengths and weaknesses of it.


## 1 Introduction

Text-to-SQL parsing, which centers on automated generation of SQL queries from natural language questions, has emerged as a significant research in both academic and industrial sectors. This long-term challenge is crucial for improving the convenience of operating databases and reducing the dependence on SQL expertise (Qin et al., 2022; Deng et al., 2022).

Recent advances in LLMs, especially those have hundreds of billions parameters, achieved significant breakthroughs in Text-to-SQL. However, these approaches based on proprietary LLMs encounter significant data privacy and cost concerns in practical applications, making them unsustainable as long-term privacy solutions. Recent studies have reported the performance of fine-tuned small LLMs that their effectiveness remains inferior to SOTA methods powered by GPT-4. For example, DAIL-SQL (Gao et al., 2024) demonstrated that the fine-tuned small LLMs struggle to learn from contextual examples due to overfitting to zero-shot prompts. MAC-SQL (Wang et al., 2024) fine-tuned Llama-7b (Meta, 2023) for multi-agent collaborative framework, it still falls short of the improved methods based on proprietary LLMs.

To alleviate the challenges, we introduce CPO-SQL, an approach aiming to efficiently boost the small LLMs for Text-to-SQL via In-**C**ontext Learning (Brown et al., 2020) and **P**reference **O**ptimization (Brown et al., 2020), as shown in Figure 1. We build the enhanced training set by sampling demonstrations from beta distribution, then fine-tune the small LLMs to empower them with Text-to-SQL ICL capability. This effectively avoids the fine-tuned small LLMs overfitting to zero-shot prompt, thereby we can leverage retrieval-augmented generation to improve its accuracy. Moreover, we enhance the small LLMs' capability in handling difficult Text-to-SQL tasks through preference optimization. We adopt an agile semi-automated process to build a new Spider preference set, which is derived from Spider training set (Yu et al., 2018b), consisting of 1388 Question-SQL pairs. Then we employ SFT-enhanced preference optimization to train the small LLMs on Spider preference set, enabling them to learn better SQL generation styles. It performs both Direct Preference Optimization and Supervised Fine-Tuning simultaneously in training process, which further boosts the performance of small LLMs in challenging Text-to-SQL tasks.

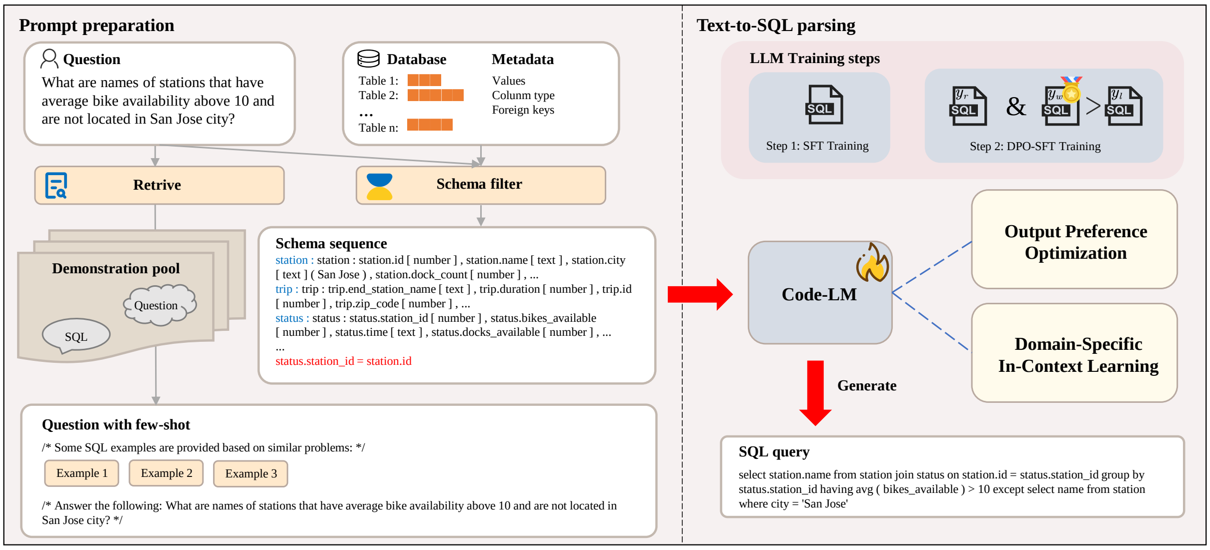


Figure 1: **An overview of CPO-SQL.** We train the code-LLM from both In-**C**ontext Learning and **P**reference **O**ptimization. The former enables it to be applied to RAG and the latter enhances it in handling difficult tasks. They efficiently bridge the gap between small LLMs and proprietary LLMs.

We evaluate the performance of our method on Spider (Yu et al., 2018b) with three sizes of Code-LLMs: DeepSeek-Coder (1.3b, 7b) (DeepSeekAI, 2024) and Starcoder2 (3b) (BigCode, 2023). The results demonstrate that our method efficiently enhanced the small LLMs' Text-to-SQL ICL ability by fine-tuning. Based on the checkpoints after fine-tuning, we perform SFT-enhanced preference optimization using elaborate Spider preference set and the non-optimized part of Spider training set. Surprisingly, despite having fewer parameters (7b) than GPT-4, the model achieved accuracy rates of 84.43% on Spider development set and 87.10% on Spider test set, reaching performance comparable to SOTA methods using GPT-4 (Li et al., 2024; Xie et al., 2024). Finally, we evaluate the robustness of our method on three Spider's variants and shed light on the strengths and weaknesses of our method.

Our contribution are threefold: (1) We introduce beta distribution sampling in similar examples matching for training set enhancement, to avoid fine-tuned small LLMs from overfitting on zero-shot prompts and leverage RAG to efficiently bridge the gap with proprietary LLMs. (2) We propose a new Spider preference set with 1388 Question-SQL pairs, constructed by an agile semi-automated process. (3) We employ SFT-enhanced preference optimization to train the small LLMs on Spider preference set. It aims to optimize the small LLMs for SQL generation in complex query scenarios and preserve adaptability while facing questions of varying difficulty.

## 2 Related work

**Text-to-SQL with LLMs** LLM-based Text-to-SQL parsing includes two paradigms: Prompting LLMs and Fine-tuning small LLMs. **Prompting methods**, as demonstrated by DIN-SQL (Pourreza and Rafiei, 2023), CoT-style (Tai et al., 2023), SQL-Prompt (Sun et al., 2023), Self-debugging (Chen et al., 2023), and DAIL-SQL (Gao et al., 2024), are tailored to guide LLMs through intricate sub-tasks such as schema linking, difficulty classification, and self-correction. They powered by advanced proprietary LLMs, such as GPT-4, raising concerns of data privacy and inference overheads. Besides, **Fine-tuning methods**, though proven effective in coding tasks, remain relatively under-explored in this field due to the expensive training overheads. Notably, DAIL-SQL has investigated fine-tuning small LLMs (e.g., LLaMA), revealing performance gaps compared to prompting proprietary LLMs. MAC-SQL (Wang et al., 2024) proposed a novel multi-agent framework for Text-to-SQL, and introduced a fine-tuned Code Llama as agents to solve the subtasks. SQL-PaLM (Google, 2024) focus on LLMs at larger scales, to investigate the potential of achieving significant gain with the increase of model size due to the emergent ability of large models. Different from previous researches, we primarily focus on enhancing the Text-to-SQL capabilities of small LLMs through instruction fine-tuning, including Text-to-SQL ICL and preference optimization, to efficiently narrow the gap with proprietary LLMs.

**Simplification of SQL** To alleviate the challenge of Text-to-SQL parsing, previous studies focused on developing a SQL intermediate representation (IR) aiming at minimizing the mismatch between natural language descriptions and their corresponding SQL queries. SyntaxSQLNet (Yu et al., 2018a), EditSQL (Zhang et al., 2019), RAT-SQL (Wang et al., 2020), and NatSQL (Gan et al., 2021c), have sought to refine IR methods by removing or combining various SQL clauses to simplify the SQL representation. These efforts narrowed the gap between natural language and SQL in semantics. Nevertheless, the IR methods require extensive manual annotation and involve intricate transformation logic. Besides, they can not fully reconstruct the SQL statements, resulting in information loss. In contrast to previous studies, our objective is to agilely construct a preference dataset that includes both the original SQL and more concise variants, enabling our model to learn better SQL generation styles from it.

## 3 Problem Definition

**Text-to-SQL Task** The Text-to-SQL task involves generating a SQL query $y$ that corresponds to a user question $\mathcal{Q}$ based on a database schema $\mathcal{S}$ , and demonstrations $\mathcal{E}$. The database schema $\mathcal{S}$ of relational database $\mathcal{D}$ includes (1) a set of tables $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_m\}$, (2) a set of columns $\mathcal{C} = \left\{\mathcal{C}^1_{\mathcal{T}_1}, ..., \mathcal{C}^n_{\mathcal{T}_1}, \mathcal{C}^1_{\mathcal{T}_2}, ..., \mathcal{C}^n_{\mathcal{T}_2}, \mathcal{C}^1_{\mathcal{T}_m}, ..., \mathcal{C}^n_{\mathcal{T}_m}\right\}$ associated with the tables, (3) and a set of foreign key relations $\mathcal{R} = \left\{\left(\mathcal{C}^i_k, \mathcal{C}^j_h\right) | \mathcal{C}^i_k, \mathcal{C}^j_h \in \mathcal{C}\right\}$ . Here, $m$ and $n$ denote the number of table names and column names, respectively. Finally, with the language model policy $\pi$, the Text-to-SQL task could be formulated as:

$$y = f\left(\mathcal{Q}, \mathcal{S}, \mathcal{E} | \pi\right), \tag{1}$$

## 4 Methodology

### 4.1 Model Overview

The framework is shown in Figure 1, utilizing a fine-tuned code-LLM as the core of Text-to-SQL parsing with a retriever to provide similar examples and a filter to build relevant schema sequence from database. Firstly, We enhance the training set to equip the fine-tuned small LLMs with domain-specific ICL ability. Next, we construct the Spider preference set and further optimize the small LLMs' capability to handle challenging tasks by SFT-enhanced preference optimization.

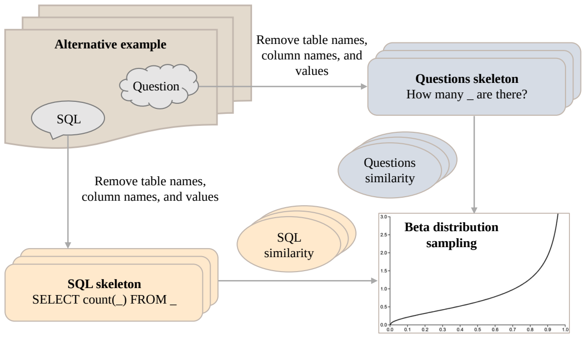


Figure 2: Examples selection for training data.

### 4.2 Text-to-SQL In-Context Learning

*Fine-tuned small LLMs fail to learn from contextual examples due to overfitting zero-shot prompts*, as mentioned by DAIL-SQL (Gao et al., 2024), is our main challenge. To solve it, we first append similar examples to the Spider training set $\mathcal{D}_{\text{train}}$. For each training data $x_i = (y_i, \mathcal{Q}_i, \mathcal{S}_i, \mathcal{K}_i)$ , we match the suitable examples among the rest of the training set $\{x_j | x_j \in \mathcal{D}_{train} - x_i\}$, by sorting them based on question similarity, then select data $x_j = (y_j, \mathcal{Q}_j)$ with SQL similarity higher than a certain threshold $\theta$ as the examples $\mathcal{E}$ for $x_i$.

However, we observe that method above leads the small LLMs to become over-dependent on the provided demonstrations when generating SQL statements. If the retrieved examples are irrelevant to the task, it can significantly affect the accuracy of the generated SQL. Besides, the examples at front of the context are always the most similar to the current task, leading the fine-tuned small LLMs to be "lazy" in context learning, that is, overly relying on $\mathcal{E}_1$ while not taking full advantage of subsequent examples, resulting in poorer performance.

**Beta Distribution Sampling** To mitigate the challenge, we introduce beta distribution sampling for examples selection, as shown in Figure 2. We view the beta distribution $X \sim Beta(\alpha, \beta)$ as the prior probability distribution for candidate example $x_j, x_j \in \mathcal{D}_{train} - x_i$ being selected as the final demonstration. We hope the candidate $x_j$ that is more similar to current data $x_i$ has a higher probability of being selected. Therefore, we sort the candidate examples $\mathcal{E}_{candidate} = \{x | x \in \mathcal{D}_{train} - x_i\}$ according to the normalized similarity $S_{x_i \sim x_j}$ between current data $x_i$ and candidate $x_j$. Then sampling $p \in (0, 1)$ from the beta distribution $X \sim Beta(\alpha, \beta)$ each time, and select the candidate $x_j$ with the minimum difference $\left|S_{x_i \sim x_j} - p\right|$ as the target example of $x_i$.

We employ beta distribution sampling strategy to select $h$ examples from $\mathcal{E}_{candidate}$ based on the question similarity, and then select $k$ examples as final $\mathcal{E}$ from $h$ candidates based on the SQL similarity using the same strategy. We extract the skeleton of SQL and mask the database content of the questions as preprocessing before similarity calculation. Besides, we measure the similarity of questions $Q$ and SQL statements $l$ by Euclidean distance and Jaccard similarity respectively.

Attributed to the uncertainty of $S_{x_i \sim x_j}$ introduced by beta distribution sampling , it effectively avoids the small LLMs overfitting to similar examples, improving their ability to learn from multiple examples. Moreover, the examples closely aligned with the current task have a greater likelihood of selection, ensuring the ICL training effectiveness.

### 4.3 Spider Preference Set

To perform preference optimization on LLMs, we describe the agile construction of the offline SQL preference set in this section. The complex Text-to-SQL tasks in Spider (Yu et al., 2018b) pose significant challenges to small LLMs. In order to reduce the difficulty of Text-to-SQL parsing, previous studies mainly focus on SQL intermediate representation (IR). They require extensive manual annotation, and the transformation logic is intricate, which cannot reconstitute SQL integrally, leading to the information loss. In contrast to these IR-based methods, we aim to efficiently construct a preference dataset that includes both the original SQL and more concise variants, then optimize the small LLMs to learn the improved SQL generation styles. We consider the following six optimizations of SQL statements, as shown in Figure 3, including **Non-essential components**, **New SQL feature**, **Table join**, **Set operation**, **Sorting operation**, and **Other optimization involved SQL skeleton**. See AppendixA for more details.

We adopt an agile semi-automated process to build the Spider preference set, as depicted in Figure 4, with the objective of enabling the small LLMs to learn the superior SQL style from it. Based on Spider training set, we feed the SQL statement $y_w$ targeted for optimization into Qwen-max (Alibaba, 2023), along with the associated database schema and question. Then we prompt the LLM to generate multiple equivalent SQL statements $y^*$. By using Test Suite (Zhong et al., 2020), we compare the execution results between SQL $y^*$ and original SQL

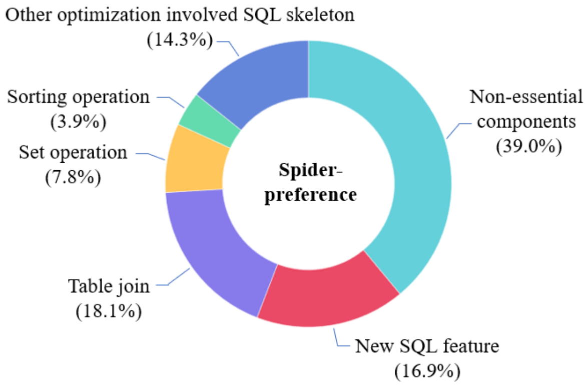


Figure 3: Composition of Spider preference set.

$y_l$, ensuring the same result of them. Finally, we appraise the SQL that have passed inspection based on six types of optimization and select the refined SQL $y_w$ along with the original SQL $y_l$ to join the preference set, finalizing the dataset construction, which consists of 1388 samples.

### 4.4 Text-to-SQL Preference Optimization

Direct Preference Optimization (DPO) (Rafailov et al., 2023), which directly optimizes the model policy based on the ideal probability distribution of human preferences without reward model, has been proven effective in text generation. However, in our attempt to perform DPO for small LLMs on Spider preference set, the SQL statements generated by the small LLMs lack logical coherence, falling short of our anticipated outcomes. We find that the lack of cross-entropy loss in DPO leads to the divergence of the models' generated results. Consequently, we propose **SFT-enhanced preference optimization** for Text-to-SQL training, which integrates cross-entropy loss into the DPO training stage to enhance it, and supports mixed training of supervised fine-tuning data and preference optimization data by modifying the loss calculation.

Our SFT-enhanced preference optimization includes three phases: 1) model initialization; 2) offline preference set construction; 3) optimize the small LLM based on preferences set.

**Model Initialization** Our model initialization is the same as DPO, where we initialize the reference model $\pi_{\text{ref}}$ with a language model $\pi^{\text{SFT}}$ generally. The language model $\pi^{\text{SFT}}$ obtained by fine-tuning a pre-train model on high-quality data specific to the downstream task, which refers to Text-to-SQL parsing in the experiment.

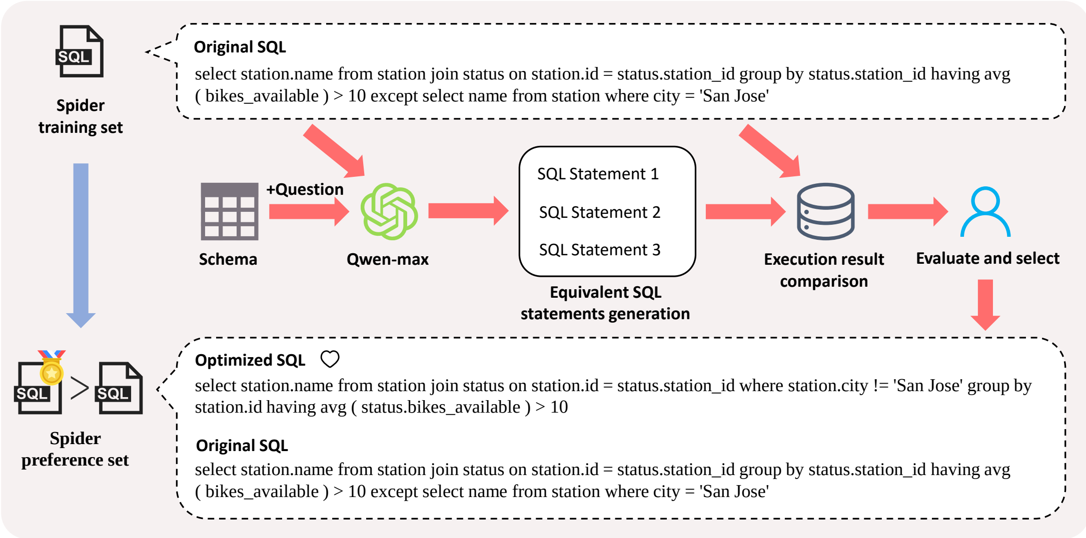


Figure 4: **The agile construction process of Spider preference set.** We provide the database schema and questions to Qwen-max, prompting it to generate equivalent SQL statements. Then we compare their execution results with the original one for filtration by test suite. Finally, we evaluate the optimization results and build the preference set.

**Preferences Set Construction** The fine-tuned model $\pi^{\mathrm{SFT}}$ is provided with prompts $x$ to generate pairs of answers $(y_1, y_2) \sim \pi^{\mathrm{SFT}}(y \mid x)$. These pairs will be presented to human labelers to construct the offline preferences set $\mathcal{D}_1 = \{x^{(i)}, y_w^{(i)}, y_l^{(i)}\}_{i=1}^{N}$. For one answer over the other, the labelers indicate the preferences, denoted as $y_w \succ y_l \mid x$, where $y_w$ and $y_l$ represent the preferred and less-preferred completions among $(y_1, y_2)$ respectively.

**Original DPO** Given $\pi_{\mathrm{ref}}$ and $\mathcal{D}$ and hyper-parameter $\beta$, we optimize the language model $\pi_\theta$ to minimize $\mathcal{L}_{\mathrm{DPO}}$, where $\beta$ controls the deviation from the base reference policy $\pi_{\mathrm{ref}}$. Usually, the model $\pi_{\mathrm{ref}}$ is the same as the initial SFT model $\pi^{\mathrm{SFT}}$. The negative log-likelihood loss of DPO can be represented as:

$$\mathcal{L}_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}_1} [\log \sigma(\eta)]. \quad (2)$$

$$\eta = \beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\mathrm{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\mathrm{ref}}(y_l \mid x)}. \quad (3)$$

**SFT-enhanced preference optimization** We integrate the cross-entropy loss, typically used in supervised training for LLMs, into the DPO optimization objective to enhance the small LLMs performance in Text-to-SQL. It can be written as:

$$\mathcal{L}_{\mathrm{SFT}}(\pi_\theta) = -\mathbb{E}_{(x,y) \sim \mathcal{D}_2} [y \log \pi_\theta(y \mid x)]. \quad (4)$$

Presently, we possess both the Spider training set and the Spider preference set we constructed in the last section. Since it derived from optimizing a subset of the training set, to avoid duplication, we consider the preference data set $\mathcal{D}_{preference}$ and non-optimized part of training set $\mathcal{D}_{rest} = \{(x, y_r) \,|\, (x, y_r) \in \mathcal{D}_{train}, x \textit{ not in } \mathcal{D}_{preference}\}$ as training data in preference optimization stage. It is remarkable that we adapt $\mathcal{D}_{rest}$ to match the format of the preference dataset to ensure uniformity in the training data format, which represented as $\mathcal{D}_{rest'} = \{(x, y_w, y_l) \,|\, y_w = y_l = y_r, (x, y_r) \in \mathcal{D}_{rest}\}$.

Based on Eq. 2 and Eq. 4 , we set the objective of the SFT-enhanced preference optimization as:

$$\mathcal{L}_{\mathrm{DPO\_SFT}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[ \begin{array}{c} \log \sigma(\eta) \\ + K_{ftx} y_w \log \pi_\theta(y_w \mid x) \end{array} \right] \quad (5)$$

where hyper-parameter $K_{ftx}$ controls the degree to which the model learns the optimal data format.

During optimization, we determine whether the current data $(x, y_w, y_l)$ comes from the preference dataset $\mathcal{D}_{preference}$ $(\mathcal{D}_1)$ or the supervised dataset $\mathcal{D}_{rest}$ $(\mathcal{D}_2)$ by comparing $y_w$ and $y_l$ for equivalence. If $y_w = y_l$ , it signifies that the data originates from the supervised dataset $\mathcal{D}_{rest}$. We disregard the DPO loss for the current data, and our optimization objective function degrades to:

$$\mathcal{L}_{\mathrm{DPO\_SFT}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} [K_{ftx} y_w \log \pi_\theta(y_w \mid x)] \quad (6)$$

According to the definition 5 6, our optimization objective is to optimize the Text-to-SQL model $\pi_\theta$ through SFT-enhanced preference optimization to learn from two diverging distributions: supervised dataset $\mathcal{D}_{rest}$ and preference dataset $\mathcal{D}_{preference}$. The training process is completed in one phase.

## 5 Experiments

We evaluate our method on Spider (Yu et al., 2018b) and its three robustness-diagnostic variants: Spider-DK (Gan et al., 2021b), Spider-Syn (Gan et al., 2021a), and Spider-Realistic (Deng et al., 2021).

**Spider** stands as the classical benchmark for Text-to-SQL tasks, comprising a training set of 7,000 samples, a dev set of 1,034 samples, and a test set of 2,147 samples, being widely used to evaluate text-to-SQL parsers across various databases, requiring models to demonstrate their adaptability to unfamiliar database structures.

**Spider-DK, Spider-Syn, Spider-Realistic** are variants derived from the Spider development set, specifically designed to mimic queries that could be posed by users in real-world scenarios. Concretely, Spider-DK incorporates domain knowledge to paraphrase questions. Spider-Syn replaces schema-related words with synonyms in questions. Spider-Realistic removes explicitly mentioned column names in questions.

**Evaluation Metrics** To assess the fine-tuned small LLMs' performance in Text-to-SQL, following Yu et al., 2018b; Zhong et al., 2020, we adopt two metrics: Exact-set-Match accuracy(EM) and Execution accuracy (EX). EM determines whether the predicted SQL query perfectly matches the Gold SQL query by converting both into a data structure (Yu et al., 2018b), while EX compares the execution outcomes of the predicted and Gold SQL queries. EX is particularly sensitive to the generated values, whereas EM is not. In practice, we combine EM and EX scores to determine the best checkpoint for small LLMs.

**Implementation Details** We utilize the cross-encoder for schema selection from RESDSQL (Li et al., 2023a), and augment the schema sequence with column types and potentially useful database content based on fuzzy matching with question. We employ Sentence-BERT (Reimers and Gurevych, 2019) for question encoding during examples' retrieval. For the core LLMs, we consider three sizes: DeekSeek Coder-(3b,7b) (DeepSeekAI, 2024), and StarCoder2 (3b) (BigCode, 2023). We train them in two stages: SFT and DPO-SFT. In the SFT stage, we select the similar examples from beta distribution with parameter $\alpha$=1.5, $\beta$=0.5 and full-training the small LLMs on Spider training set with $k$-shot, $k \in [0,4]$. We specify bs=96, lr=1e-5, and employ AdamW optimizer (Loshchilov and Hutter, 2019) with linear warm-up (the first 10% training steps) and cosine decay to adjust the learning rate. In the DPO-SFT stage, we maintain the same batch size, learning rate, and optimizer as the previous stage. Differently, we fine-tune the small LLMs with QLoRA to reduce GPU memory usage. We set beam size of 8 for inference in both stages.

**Environments** We conduct all experiments on a server with 4×V100 (32GB) GPUs and 200GB of memory. Besides, we utilize DeepSpeed ZeRO-2 to mitigate the memory and compute demands of each GPU utilized for training.

### 5.1 Evaluation on In-Context Learning

In few-shot scenario, we evaluate the small LLMs under two example selection strategies: **Precise Matching** (PM) and **Beta distribution Sampling** (BS) . Under the PM strategy, we always select the examples that are most similar to the current training data as additional context. In contrast, with the BS strategy, we select the examples to be appended to the context from a beta distribution based on the normalized similarity between examples and the current training data. To ensure a fair evaluation on Spider development set, we select the best-performing checkpoint of each model. For inference, we always select the most similar example for current task.

Figure 5 reports the EM and EX results on Spider under two example selection strategies for different small LLMs. We observe that LLMs trained with BS can benefit from more contextual examples than PM. It's evident in the performance of DeepSeek Coder (1.3b) at 2-shot and StarCoder2 (3b) at 3-shot. As the number of examples increases beyond 1-shot, these LLMs consistently outperform their counterparts trained with PM in terms of both EM and EX results, which indicates that the beta distribution sampling for similar examples can prevent small LLMs from overfitting to the similar prompts provided in training. However, it should be noted that compared to GPT-4, as reported by DAIL-SQL (Gao et al., 2024), where EX increases

| Stage | Model | Size | Easy | | Medium | | Hard | | Extra | | All | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | EM% | EX% | EM% | EX% | EM% | EX% | EM% | EX% | EM% | EX% |
| SFT | DeepSeek Coder | 1.3b | 87.90 | 91.93 | 79.82 | 87.89 | 59.77 | 71.83 | 54.21 | 61.44 | 74.27 | 81.91 |
| | | 7b | 94.75 | 96.37 | 86.99 | 92.15 | 86.99 | 75.86 | 58.43 | 63.85 | 81.43 | 85.88 |
| | StarCoder2 | 3b | 92.33 | 93.14 | 82.06 | 86.54 | 60.91 | 72.41 | 45.18 | 54.21 | 75.04 | 80.56 |
| DPO-SFT | DeepSeek Coder | 1.3b | 91.12 | 94.75 | 81.83 | 88.34 | 65.51 | 77.58 | 46.98 | 58.43 | 75.72 | 83.26 |
| | | 7b | 94.75 | 95.96 | 84.52 | 91.92 | 70.68 | 78.73 | 51.20 | 63.25 | 79.30 | 86.07 |
| | StarCoder2 | 3b | 92.33 | 92.74 | 81.83 | 86.77 | 59.19 | 70.11 | 47.59 | 57.22 | 75.04 | 80.65 |

Table 1: Performance of small LLMs at SFT stage and DPO-SFT stage, across difficulty levels on the Spider's dev set. Darker shadows indicate poorer performance.

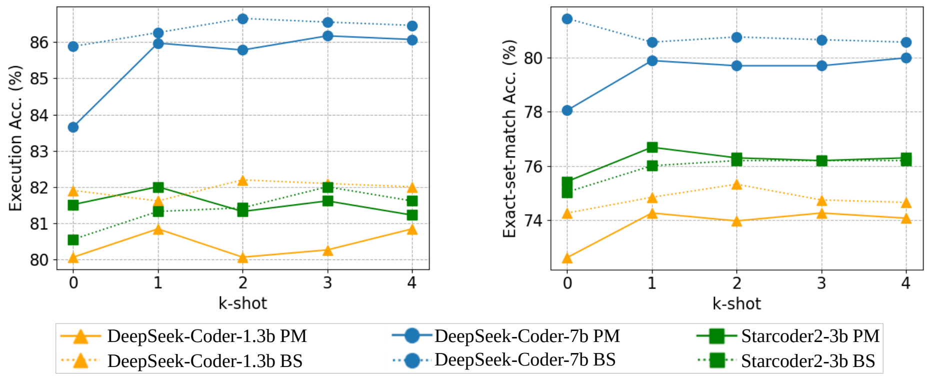


Figure 5: Few-shot evaluation with fine-tuned small LLMs on Spider-dev.

from 72.3% (0-shot) to 82.4% (5-shot), fine-tuned small LLMs' improvement gained from retrieval-augmented generation is less pronounced ($< 2.5\%$). One potential reason is that instruction fine-tuning significantly enhances the model's ability to solve problems in zero-shot scenarios, thereby narrowing the performance gap compared to scenarios where examples are provided.

### 5.2 Evaluation on Preference Optimization

As shown in Table 1, we compare the result of small LLMs from two training stages on Spider development set. The small LLMs at SFT stage are trained with the BS strategy, while those at DPO-SFT stage are optimized from LLMs at SFT stage on Spider preference set. Since the preference set introduces SQL features that are not present in the original Spider set, leading to inaccurate retrieval, we only focus on zero-shot evaluation. The results demonstrate that our SFT-enhanced preference optimization resulted in an EX improvements of 1.35% for Deepseek Coder (1.3b), 0.19% for Deepseek Coder (7b), and 0.09% for StarCoder2 (3b). This suggests that our method performs more effectively on small LLMs with fewer parameters, as our Spider preference set primarily focuses on simplifying challenging SQL that small LLMs struggle to generate.

Based on the difficulty-level stratification results, small LLMs at DPO-SFT stage demonstrate more advantages for **Medium** and **Hard** level compared to **Easy** and **Extra Hard** level. Note that small LLMs at DPO-SFT stage sometimes have a lower EM result than their corresponding models at SFT stage but a higher EX result. This occurs because EM requires strict adherence to SQL formatting, whereas LLMs at DPO-SFT stage may generate SQL statements with formats that are inconsistent with the target but yield the same execution results.

### 5.3 Result on Spider

Table 2 reports the results on Spider. Our top-performing model, DeepSeek Coder (7b) at DPO-SFT stage, achieved 87.1% EX on the test set, reaching performance comparable to SOTA methods using GPT-4 (Xie et al., 2024). This demonstrates the high-efficiency of our method. Besides, the DeepSeek Coder (1.3b) at DPO-SFT stage, which achieved 75.7% EM and 83.2% EX, stands as the best-performing model at equivalent scale, suggesting that our SFT-enhanced preference optimization effectively mitigates the challenges faced by small LLMs in Text-to-SQL parsing. The DeekSeek Coder (7b) at SFT stage also achieved commendable result (EM 76.9%, EX 86.6%), by leveraging RAG (3-shot) to learn from examples.

| Approach | Zero-Shot | Few-Shot | Fine-tuning | Dev Set | | Test Set | |
|---|---|---|---|---|---|---|---|
| | | | | EM% | EX% | EM% | EX% |
| DAIL-SQL + GPT-4 + SC (Gao et al., 2024) | | ✓ | | 68.7 | 83.6 | 66.0 | 86.6 |
| MAC-SQL + GPT-4 (Wang et al., 2024) | | ✓ | | 63.2 | **86.7** | - | 82.8 |
| DEA-SQL + GPT-4 (Xie et al., 2024) | | ✓ | | - | 85.4 | - | **87.1** |
| C3 + ChatGPT + Zero-Shot (Dong et al., 2023) | ✓ | | | 71.4 | 81.8 | - | 82.3 |
| ChatGPT (Liu et al., 2023) | ✓ | | | 34.6 | 74.4 | - | - |
| GPT-4 (OpenAI, 2024) | ✓ | | | 22.1 | 72.3 | - | - |
| T5-3B + PICARD (Scholak et al., 2021) | | | ✓ | 75.5 | 79.3 | 71.9 | 75.1 |
| Graphix-T5-3B + PICARD (Li et al., 2023b) | | | ✓ | 77.1 | 81.0 | 74.0 | 77.6 |
| RESDSQL-3B + NatSQL (Li et al., 2023a) | | | ✓ | 80.5 | 84.1 | 72.0 | 79.9 |
| SQL-PaLM (Google, 2024) | | | ✓ | 78.2 | 82.8 | - | - |
| SFT DeepSeek Coder-1.3b | | ✓ | ✓ | 74.7 | 82.1 | 69.5 | 81.2 |
| SFT DeepSeek Coder-7b | | ✓ | ✓ | **80.6** | 86.5 | **76.9** | 86.6 |
| SFT StarCoder2-3b | | ✓ | ✓ | 76.2 | 82.0 | 74.2 | 82.9 |
| DPO-SFT DeepSeek Coder-1.3b | | | ✓ | 75.7 | 83.2 | 71.0 | 81.0 |
| DPO-SFT DeepSeek Coder-7b | | | ✓ | 77.5 | 84.4 | 74.9 | **87.1** |
| DPO-SFT StarCoder2-3b | | | ✓ | 75.0 | 80.6 | 73.9 | 82.4 |

Table 2: Exact-set-Match accuracy (EM) and Execution accuracy (EX) results on Spider's development set and test set. We compare our approach with other baseline methods.

| Approach | Spider-Syn | | Spider-Realistic | | Spider-DK | |
|---|---|---|---|---|---|---|
| | EM% | EX% | EM% | EX% | EM% | EX% |
| T5-3B + PICARD (Scholak et al., 2021) | 61.8 | 69.8 | 61.7 | 71.4 | — | 62.5 |
| RESDSQL-3B + NatSQL (Li et al., 2023a) | 66.8 | **76.9** | 70.1 | 81.9 | 53.3 | 66.0 |
| ChatGPT (Liu et al., 2023) | 48.5 | 58.6 | 49.2 | 63.4 | — | 62.6 |
| SQL-Palm (Few-shot) (Google, 2024) | 67.4 | 74.6 | 72.4 | 77.6 | — | 66.5 |
| SQL-Palm (Fine-tuned) (Google, 2024) | 66.4 | 70.9 | 73.2 | 77.4 | — | 67.5 |
| SFT DeepSeek Coder-1.3b | 53.7 | 65.8 | 65.3 | 72.6 | 54.9 | 62.4 |
| SFT DeepSeek Coder-7b | **68.6** | 75.6 | **78.1** | 81.4 | **62.9** | **70.4** |
| SFT StarCoder2-3b | 60.2 | 67.4 | 72.4 | 75.1 | 56.2 | 62.2 |
| DPO-SFT DeepSeek Coder-1.3b | 62.0 | 72.2 | 66.3 | 76.7 | 52.7 | 62.9 |
| DPO-SFT DeepSeek Coder-7b | 67.4 | 74.9 | 75.0 | **82.2** | 61.1 | 68.2 |
| DPO-SFT StarCoder2-3b | 60.1 | 67.1 | 71.4 | 74.8 | 56.2 | 62.4 |

Table 3: Evaluation results on Spider-DK, Spider-Syn, and Spider-Realistic.

### 5.4 Results on Robustness Settings

Previous researches (Gan et al., 2021a; Deng et al., 2021) highlight the fragility of neural Text-to-SQL parsers faced with perturbations in questions. This fragility arises from the removal or substitution of explicitly mentioned schema items with semantically consistent words, such as synonyms, which complicates the schema linking. To investigate the robustness of the fine-tuned small LLMs, we evaluated them on three challenging variants of the Spider dataset: Spider-DK, Spider-Syn, and Spider-Realistic.

As shown in Table 3, our fine-tuned LLMs achieved SOTA performance on multiple robustness metrics: Spider-Syn (EM 68.6%), Spider-Realistic (EM 78.1%, EX 82.2%), and Spider-DK (EM 62.9%, EX 70.4%), demonstrating the high-efficiency of In-Context Learning and Preference Optimization we proposed. However, it should be noted that the improved metrics are primarily attributed to the LLMs at SFT stage, whereas LLMs at DPO-SFT stage are further optimized, did not exhibit superior robustness. We think this discrepancy may stem from redundant learning on similar data and plan to investigate it in future research.

### 5.5 Conclusion

In this paper, we propose CPO-SQL, an approach aiming to boost the Text-to-SQL capability of small LLMs via In-**C**ontext Learning and **P**reference **O**ptimization efficiently. In order to prevent the fine-tuned small LLMs from overfitting to zero-shot prompts, we enhance the training set by sampling demonstrations from beta distribution, then fine-tune the small LLMs to empower them with Text-to-SQL ICL capability. Moreover, we enhance the small LLMs' ability in handling difficult Text-to-SQL tasks through SFT-enhanced preference optimization on Spider preference set, constructed by an agile semi-automated process. Our models achieve state-of-the-art performance across multiple metrics on Spider and its three robustness-diagnostic variants, demonstrating the high-efficiency of CPO-SQL in bridging the gap between small LLMs and proprietary LLMs.

## Limitations

**Prompt engineering** We did not extensively engineer the prompts for small LLMs input, which may not be optimal in the experiments.

**Preferences Set Construction** The construction of Spider preference set only consider the SQL simplification to reduce the difficulty of Text-to-SQL parsing for small LLMs, We do not consider the execution efficiency in the real scenario and plan to explore it in future research.

**Differences in Pre-training Corpora** We focus on boosting the capability of small LLMs in Text-to-SQL via efficient fine-tuning and without considering the differences in pre-training stage. We evaluate two types of code-LLMs in the experiment: DeekSeek Coder (DeepSeekAI, 2024) and StarCoder2 (BigCode, 2023). The different proportion of programming languages in their pretraining corpus may affect our comparison of small LLMs with different sizes. For example, StarCoder2 (3b) underperforms DeepSeek Coder (1.3b) in some metrics may due to the smaller proportion of SQL in its pretraining corpus.

## Ethics Statement

**Data Disclaimer** We build a new SQL preference set based on Spider, a dataset widely used by academics and accessible to the public. Therefore, our proposed dataset does not involve any sensitive information that may harm others.

**Human Labeler** When recruiting labelers for this study, we ensure that all potential labelers are free to choose whether they want to participate and can withdraw from the study anytime without any negative repercussions. Thus, the establishment of our dataset complies with ethics.

## References


Alibaba. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

BigCode. 2023. Starcoder: may the source be with you! *Preprint*, arXiv:2305.06161.

Tom B. Brown, Benjamin Mann, and Nick Ryder et al. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *Preprint*, arXiv:2304.05128.

DeepSeekAI. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent advances in text-to-SQL: A survey of what we have and what we expect. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 2166–2187, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350, Online. Association for Computational Linguistics.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot text-to-sql with chatgpt. *Preprint*, arXiv:2307.07306.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-SQL models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2505–2515, Online. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-SQL generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John Drake, and Qiaofu Zhang. 2021c. Natural SQL: Making SQL easier to infer from natural language specifications. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2030–2042, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Google. 2024. Sql-palm: Improved large language model adaptation for text-to-sql (extended). *Preprint*, arXiv:2306.00739.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13076–13084.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024. Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *Preprint*, arXiv:2403.09732.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *Preprint*, arXiv:2303.13547.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019*.

Meta. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Preprint*, arXiv:2304.11015.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *Preprint*, arXiv:2208.13629.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741. Curran Associates, Inc.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ruoxi Sun, Sercan Arik, Rajarishi Sinha, Hootan Nakhost, Hanjun Dai, Pengcheng Yin, and Tomas Pfister. 2023. SQLPrompt: In-context text-to-SQL with minimal labeled data. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 542–550, Singapore. Association for Computational Linguistics.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain of thought style prompting for text-to-SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5376–5393, Singapore. Association for Computational Linguistics.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. Mac-sql: A multi-agent collaborative framework for text-to-sql. *Preprint*, arXiv:2312.11242.

Yuanzhen Xie, Xinzhou Jin, Tao Xie, MingXiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, ChengXiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for enhancing attention: Improving llm-based text-to-sql through workflow paradigm. *arXiv preprint arXiv:2402.10671*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018a. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5338–5349, Hong Kong, China. Association for Computational Linguistics.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411, Online. Association for Computational Linguistics.

# A SQL Optimization Categories

## Non-essential components

```
Original SQL:
select distinct ( _ ) from _ | select distinct ( catalog_entry_name ) from
    catalog_contents

Optimized SQL:
select distinct _ from _ | select distinct catalog_entry_name from catalog_contents
```

## New SQL feature

```
Original SQL:
select distinct _ from _ where _ | select distinct department.creation from
    department join management on department.department_id = management.
    department_id join head on management.head_id = head.head_id where head.
    born_state = 'Alabama'

Optimized SQL:
select _ from _ using ( _ ) join _ where _ group by _ | select department.creation
    from department join management using ( department_id ) join head on management.
    head_id = head.head_id where head.born_state = 'Alabama' group by department.
    creation
```

## Table join

```
Original SQL:
select _ from _ where _ | select candidates.candidate_id from people join candidates
     on people.person_id = candidates.candidate_id where people.email_address = '
    stanley.monahan@example.org'

Optimized SQL:
select distinct _ from _ where _ in ( select _ from _ where _ ) | select distinct
    candidate_id from candidates where candidate_id in ( select person_id from
    people where email_address = 'stanley.monahan@example.org' )
```

## Set operation

```
Original SQL:
select _ from _ where _ | select candidates.candidate_id from people join candidates
     on people.person_id = candidates.candidate_id where people.email_address = '
    stanley.monahan@example.org'

Optimized SQL:
select distinct _ from _ where _ in ( select _ from _ where _ ) | select distinct
    candidate_id from candidates where candidate_id in ( select person_id from
    people where email_address = 'stanley.monahan@example.org' )
```

## Sorting operation

```
Original SQL:
select _ from _ order by _ desc limit _ | select acc_percent from basketball_match
    order by acc_percent desc limit 1

Optimized SQL:
select max ( _ ) from _ | select max ( basketball_match.acc_percent ) from
    basketball_match
```

**Other optimization involved SQL skeleton**.

```
Original SQL:
select _ from _ except select _ from _ | select customer_name from customers except
    select customers.customer_name from customers join first_notification_of_loss on
     customers.customer_id = first_notification_of_loss.customer_id

Optimized SQL:
select _ from _ where _ not in ( select _ from _ ) | select customer_name from
    customers where customer_id not in ( select customer_id from
    first_notification_of_loss )
```

# B Prompt details

## B.1 Few-shot Prompt

```
/* Some SQL examples are provided based on similar problems: */
/* Answer the following: What is the average and minimum age of all artists from
    United States. */
select avg ( _ ) , min ( _ ) from _ where _ | select avg ( age ) , min ( age ) from
    artist where country = 'United States'

/* Answer the following: What is the average distance and average price for flights
    from Los Angeles. */
select avg ( _ ) , avg ( _ ) from _ where _ | select avg ( distance ) , avg ( price
    ) from flight where origin = 'Los Angeles'

/* Answer the following: What is the average and maximum number of total passengers
    for train stations in London or Glasgow? */
select avg ( _ ) , max ( _ ) from _ where _ | select avg ( total_passengers ) , max
    ( total_passengers ) from station where location = 'London' or location = '
    Glasgow'

/* Given the following database schema: */
country : country.surfacearea [ number ] , country.population [ number ] , country.
    continent [ text ] ( North America ) , country.region [ text ] ( North America )
     , country.name [ text ] | city : city.countrycode [ text ] ( ARE , NOR ) , city
    .population [ number ] , city.id [ number ] , city.name [ text ] ( Americana ,
    Northampton ) , city.district [ text ] ( Northern ) | sqlite_sequence :
    sqlite_sequence.name [ text ] , sqlite_sequence.seq [ text ] | countrylanguage :
     countrylanguage.countrycode [ text ] ( ARE , NOR ) , countrylanguage.language [
     text ] ( Northsotho ) , countrylanguage.percentage [ number ] , countrylanguage
    .isofficial [ text ] | city.countrycode = country.code | countrylanguage.
    countrycode = country.code

/* Answer the following: What is the total population and average area of countries
    in the continent of North America whose area is bigger than 3000 ? */


/* Expected output */
select sum ( _ ) , avg ( _ ) from _ where _ | select sum ( population ) , avg (
    surfacearea ) from country where continent = 'north america' and surfacearea >
    3000
```

### B.2 Prompt for generating SQL variants

```
/* Given the following database schema: */
station : station.id [ number ] , station.name [ text ] , station.city [ text ] (
    San Jose ) , station.dock_count [ number ] , station.long [ number ] | weather :
     weather.mean_temperature_f [ number ] , weather.max_gust_speed_mph [ number ] ,
     weather.min_temperature_f [ number ] , weather.max_wind_speed_mph [ number ] ,
    weather.max_dew_point_f [ number ] | trip : trip.end_station_name [ text ] ,
    trip.duration [ number ] , trip.id [ number ] , trip.zip_code [ number ] , trip.
    end_date [ text ] | status : status.station_id [ number ] , status.
    bikes_available [ number ] , status.time [ text ] , status.docks_available [
    number ] | status.station_id = station.id

/* Answer the following: What are names of stations that have average bike
    availability above 10 and are not located in San Jose city? */

/* SQL statement */
select station.name from station join status on station.id = status.station_id group
     by status.station_id having avg ( bikes_available ) > 10 except select name
    from station where city = 'San Jose'

Please rewrite the SQL statement above based on the database schema and question.
    You should follow these rules:
1.Ensuring that the rewritten SQL statement is equivalent to the original SQL
    statement.
2.Try to make the sql statements more concise than the original ones.
3.Don't use aliases in SQL statements.
4.Please directly output sql statements.

Output format:
#1
```sql
SELECT ...
```
#2
```sql
SELECT ...
```
#3
```sql
SELECT ...
```
"""
```